PegaRULES Process Commander v5.4
Enterprise Class Structure

Best Practices and Implementation Guidelines

August 17, 2009

© Copyright 2009
Pegasystems Inc., Cambridge, MA
All rights reserved.

This document describes products and services of Pegasystems Inc. It may contain trade secrets and proprietary information. The document and product are protected by copyright and distributed under licenses restricting their use, copying, distribution, or transmittal in any form without prior written authorization of Pegasystems Inc.

This document is current as of the date of publication only. Changes in the document may be made from time to time at the discretion of Pegasystems. This document remains the property of Pegasystems and must be returned to it upon request. This document does not imply any commitment to offer or deliver the products or services provided.

This document may include references to Pegasystems product features that have not been licensed by your company. If you have questions about whether a particular capability is included in your installation, please consult your Pegasystems service consultant.

Although Pegasystems Inc. strives for accuracy in its publications, any publication may contain inaccuracies or typographical errors. This document or Help System could contain technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Pegasystems Inc. may make improvements and/or changes in the information described herein at any time.

For Pegasystems trademarks and registered trademarks, all rights are reserved. Other brand or product names are trademarks of their respective holders.

This document is the property of:
Pegasystems Inc.
101 Main Street
Cambridge, MA 02142-1590

Phone: (617) 374-9600
Fax: (617) 374-9620
www.pega.com

Contents

1 Overview 1

1.1 PRPC Base Product Layer 2

1.2 Enterprise Reuse and Divisional Reuse Organizational Layers 2

1.3 Framework Layer 4

1.4 Implementation Layer 6

2 Changes and Enhancements in PRPC v5.4 that affect the ECS 8

3 Implementing Applications Based on The Enterprise Class Structure 11

3.1 In Which Layer Do You Put New Rules 11

3.2 What To Do if a Rule is Put into the Wrong Layer 12

3.3 Calling Flows in the Framework Layer 12

3.4 Activities 12

3.5 Connector Activities and Connect Rules 12

3.6 Services 13

3.6.1 Service Package 13

3.6.2 Service Rule 13

3.6.3 Deployment 13

3.7 How to Test the CBF 14

3.8 Versioning of External Services (Any Protocol) 14

3.9 Specialization of a CBF 14

3.10 Making a Single Work Object Available as Cover or Object 15

3.11 RuleSets 15

3.12 Applications 16

3.13 Access Roles 16

3.14 Product and Product Patch 17

4 Appendix A: Compatibility of Applications and Frameworks built on the pre-5.4 Enterprise Class Structure 18

4.1 Applications 18

4.2 Customer-Built Frameworks (CBFs) 18

4.3 Extensions 18

4.4 Data and Interface Classes 18

5 Appendix B: Classes & RuleSets generated by the Enterprise Application Accelerator 20

1 Overview

The class hierarchy is an integral part of every application. Designing a scalable, extensible enterprise class structures from the beginning of your project helps avoid costly re-factoring. The enterprise class structure enables PRPC applications to co-exist with each other and with customer or Pega built frameworks.

The enterprise class structure is also the foundation for enterprise reuse. Asset reuse is integral to the long-term success of BPM within the organization and ultimately, the reusability requirements drive the selection of your class hierarchy design.

Figure 1 below shows the Pegasystems recommended Enterprise Class Structure for applications built on the PegaRULES Process Commander v5.4 platform.

> ***Note:*** *This class structure is automatically generated by the Enterprise Application Accelerator (EAA) wizard available in PRPC v5.4 when you choose the 'New Framework and Implementation' option.*

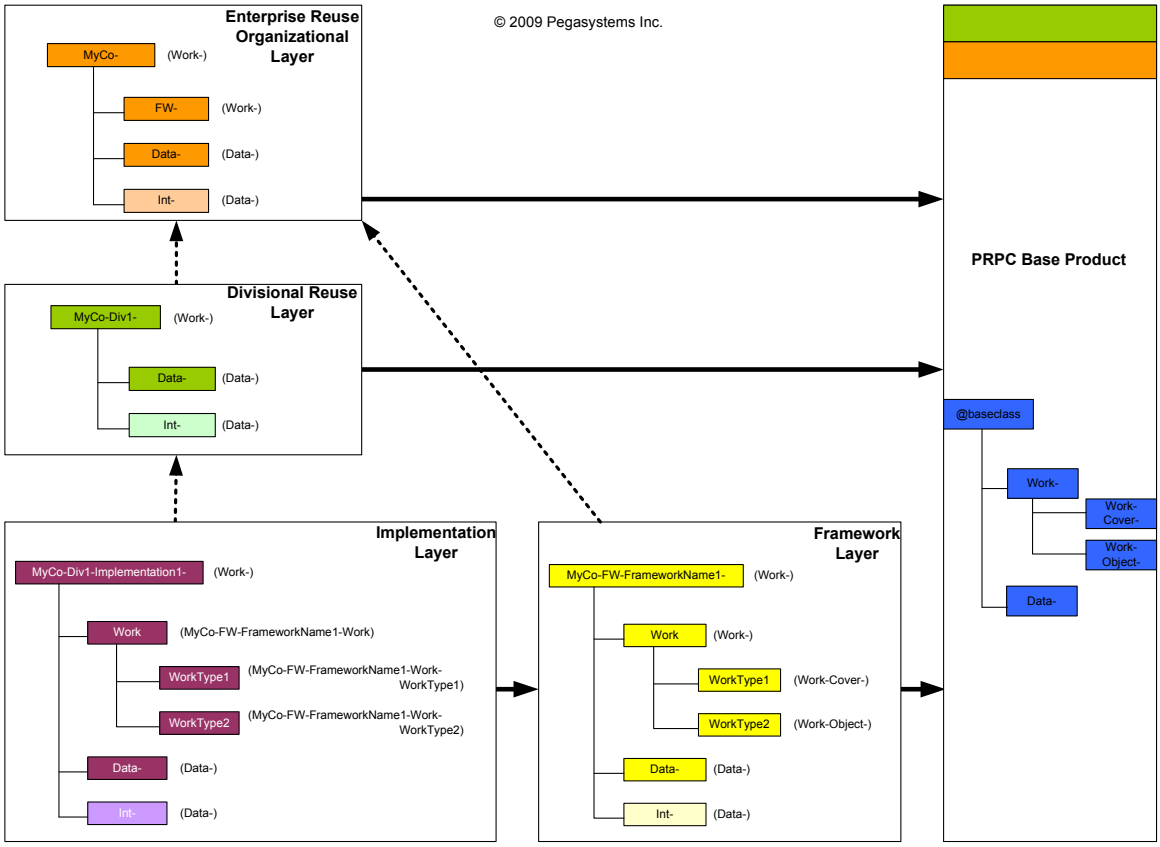

Figure 1: PRPC 5.4 Best Practice Enterprise Class Hierarchy

1.1 PRPC Base Product Layer

The PRPC Base Product Layer consists of the Pega-4 RuleSets and its classes and rules. Loaded as part of the PRPC platform install, this layer contains all rules necessary for workflow processing and other areas of Process Commander. While the Pega-4 RuleSets are locked, rules that apply to many of the Pega classes (Work-, Embed-, Data-, @baseclass, etc.) can be defined in application RuleSets to support enterprise-wide reuse.

Do not attempt to override standard activities, harnesses, flow actions, etc. that may or may not be marked as final rules. This will ensure a smoother migration to new releases of the PRPC platform.

1.2 Enterprise Reuse and Divisional Reuse Organizational Layers

The Enterprise Reuse and Divisional Reuse Organizational Layers is where extensions to the work and data classes exist. This may include enterprise wide or division wide business logic (standard properties, flows, decision tables, models, SLA rules, etc.), data classes (such as generated class structures for connector request/response messages) as well as other extensions that exist.

The classes within the Divisional Reuse layer inherit from their counterparts in the Enterprise Reuse layer; the classes of the Enterprise Reuse layer inherit directly from the Work- and Data- classes.

ENTERPRISE REUSE LAYER:

MyCo- (Work-)
Enterprise Reuse layer base class. This class is typically generated by the Setup Organization step which is part of the PRPC installation. The class record must be modified to have Work- as the directed parent. This class serves as namespace qualifier as well as an extension class for enterprise wide work rules (standard properties, flows, sections, SLAs, etc.)

MyCo-FW- (Work-)
Namespace qualifier for Customer Built Frameworks (CBFs).

MyCo-Data- (Data-)
Namespace qualifier for enterprise wide data classes available to all applications and frameworks.

MyCo-Int- (Data-)
Namespace qualifier for enterprise wide integration classes (generated data classes for connector rules) available to all applications and frameworks.

DIVISIONAL REUSE LAYER:

MyCo-Div1- (Work-)
Divisional Reuse layer base class. The first such division (and corresponding class) is generated by the Setup Organization step which is part of the PRPC installation. The class record must be modified to have Work- as the directed parent. This class serves as an extension class for work rules shared across the division (standard properties, flows, sections, SLAs, etc.)

Create one Divisional Reuse base class per division that implements one or more of the CBFs from the Framework layer.

MyCo-Div1-Data- (Data-)
Parent class for division wide data classes available to all applications and frameworks.

MyCo-Div1-Int- (Data-)
Parent class for division wide integration classes (generated data classes for connector rules) available to all applications and frameworks.

Table 1: The Classes of the Enterprise Reuse and Divisional Reuse Organizational Layers

1.3 Framework Layer

The Framework Layer stores base rules, processes, and components for a given application. A customer built framework (CBF) serves as the basis for N number of deployments throughout the enterprise.

A CBF must be able to stand on its own; therefore, this layer contains all rules necessary to implement its functionality. While there will be a class group and its mapping to a database table for testing purposes, the actual mapping of the application to production database table occurs on the Implementation layer. Organization specific logic, processes and rules must be placed on the Implementation layer as well; only default edits, values, and application structures have their home in the CBF layer. The CBF and its implementation classes can be considered the equivalent of an "ASP" model, where the CBF is the base application.

Pegasystems-built frameworks also reside in the Framework layer. A customer application that is built on a Pegasystems solution framework may directly implement the solution framework classes, or they may be a CBF themselves. In the former case, there are one or more specialization layers on the Implementation level, and the Pega solution framework on the Framework level. In the latter case, there are one or more specialization layers on the Implementation level and the CBF and the Pega solution framework on the Framework level. The CBF has the Pega framework as its directed parent.

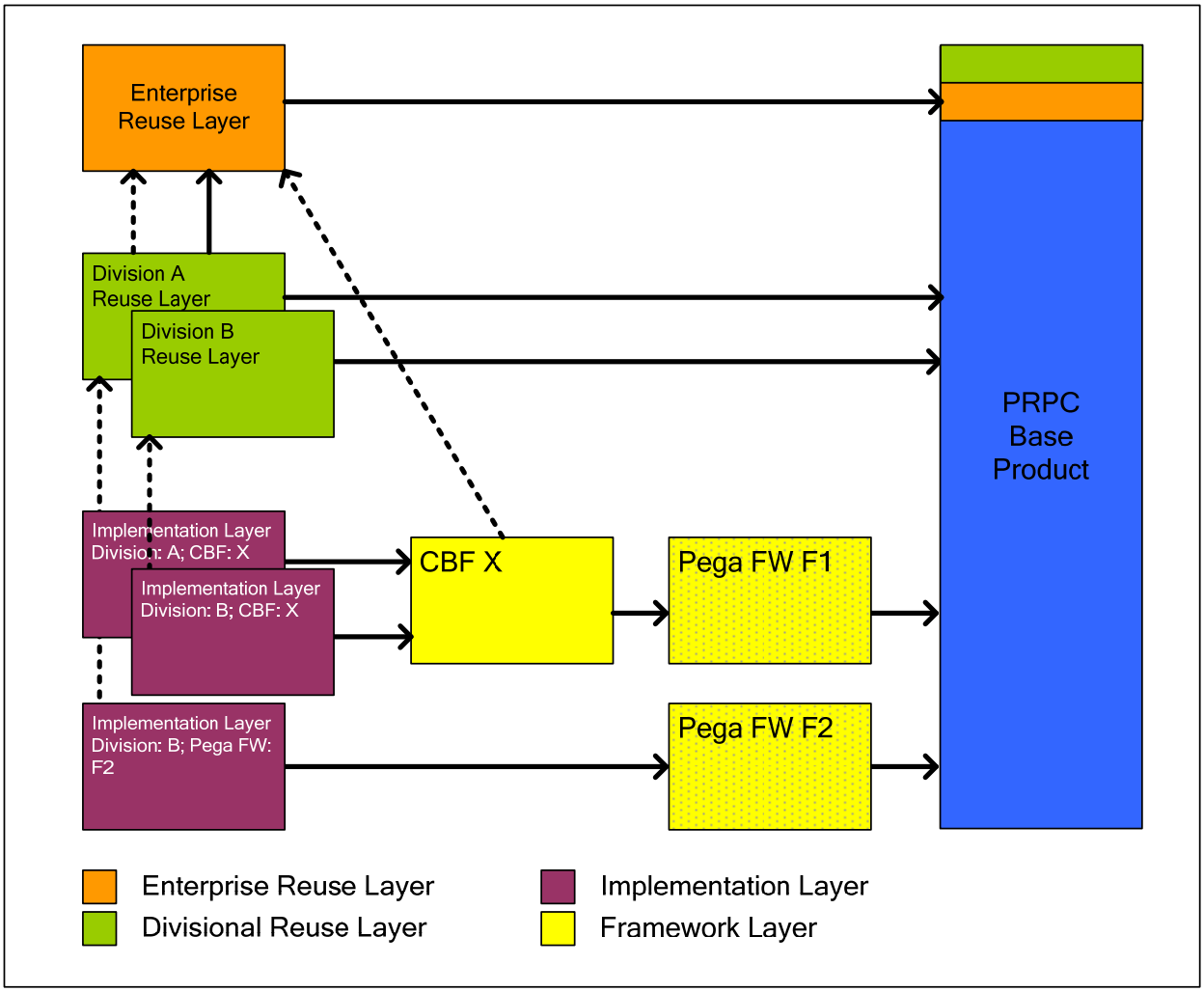

Figure 2: Pegasystems Solution Frameworks – implemented with (F1) and without (F2) a CBF

MyCo-FW- (Work-)
Namespace qualifier for all Customer Built Frameworks (CBFs).

MyCo-FW-FrameworkName1- (Work-)
Namespace qualifier for a specific CBF by the name of FrameworkName1.

MyCo-FW-FrameworkName1-Work (Work-)
Concrete class serving as the class group for the CBF, mapping instances of the CBF work classes to a database table. (This mapping is for testing purposes only; the mapping of the actual work objects happens on the Implementation layer – see below.)

MyCo-FW-FrameworkName1-Work-WorkType1 (Work-Cover-)
MyCo-FW-FrameworkName1-Work-WorkType2 (Work-Object-)
CBF work classes that implement cover, folders and/or work objects. These classes directly inherit from the Work-Cover- and Work-Object- classes in the PRPC Base Product layer.

MyCo-FW-FrameworkName1-Data- (Data-)
Parent class for data classes specific to the framework.

MyCo-FW-FrameworkName1-Int- (Data-)
Parent class for generated data classes (for connector rules) specific to the framework.

Table 2: The Classes of the Framework Layer

1.4 Implementation Layer

The Implementation layer contains the rules necessary to implement a Pega built framework or Customer Built Framework (CBF). This layer contains any specialization rules as well as the application class group(s) of the framework for the organization and division that it belongs to. The layer also contains flow shells and/or service activities used to start their generalized counterparts in the Framework layer.

MyCo-Div1-Implementation1- (Work-)
This abstract class serves as namespace qualifier and the organizational pattern inheritance path.

MyCo-Div1-Implementation1-Work (MyCo-FW-FrameworkName1-Work)
Concrete class serving as the class group for the application, mapping the instances of the application work classes to the production database table.

MyCo-Div1-Implementation1-Work-WorkType1 (MyCo-FW-FrameworkName1-Work-WorkType1)
MyCo-Div1-Implementation1-Work-WorkType2 (MyCo-FW-FrameworkName1-Work-WorkType2)
Application specific implementation work classes of Work Covers, Folders or Objects. They directly inherit from the respective CBF classes in the Framework layer.

MyCo-Div1-Implementation1-Data- (Data-)
Abstract parent class for application specific data classes.

MyCo-Div1-Implementation1-Int- (Data-)
Abstract parent class for generated data classes (for connector rules) specific to the application.

Table 3: The Classes of the Implementation Layer

2 Changes and Enhancements in PRPC v5.4 that affect the ECS

Changes and enhancement in version PRPC v5.4 of the PRPC base product affecting enterprise reuse and the recommended Enterprise Class Structure include:

1. The Work- class has been “opened” to application RuleSets, i.e. it is now possible to save rules that apply to the Work- class or its named descendents to application RuleSets. However, it is **recommended that you do not save** enterprise wide and division wide extensions to work classes directly in the Work- class. The reason is that burgeoning numbers of rules in that class would “pollute” it. Because the Class Explorer tool shows all inherited rules of application work classes, navigation would become difficult as the number of rules in the Work- parent increases.

   Instead, extensions to the work class should be saved in the Enterprise Reuse or Divisional Reuse layers, based on their scope. Enterprise wide extensions to work classes should be saved in MyCo-; extensions for reuse within a single division should be stored in MyCo-Div1-.

2. RuleSets can now be categorized into the following RuleSet types:
   - `Standard` — A "normal" RuleSet that can contain all types of rules. (All RuleSets created in PRPC v5.3 and earlier belong to this category.)
   - `Component` — Contains rules that are designed to define reusable applications or functionality that executes embedded within an object. When a component RuleSet is installed in multiple systems, the class of the object may vary from system to system.
   - `Shared` — Contains a small number of rules that each operate on common top-level page of a single class (or of subclasses of that class).
   - `Override` — Contains rules that by design override rules in other, non-override RuleSets. Override RuleSet versions appear at the top of the RuleSet list and during rule resolution supersede any rules with identical names found in RuleSet versions lower in your RuleSet list. (Suggested uses include enterprise-wide information, such as a company logo image in a binary file rule. A retail chain that operates many stores in multiple states can create an Override rule to force the sales price of an item to a uniform value company wide.)

3. The scope of Application rules has been extended:
   - The Application rule form now contains separate sections for Application RuleSets (type Standard) and Component and Shared RuleSets.
   - A Customization RuleSets section allows specifying candidates for Production RuleSets in access groups that reference this Application.
   - A Special Purpose RuleSets section allows you to list one or more Override type RuleSets that are moved to the top of the requestor’s RuleSet stack.

- The *Details* tab allows capturing business objectives, work types, actors and categories of supporting use cases.
- The *Use Case* tab lists all Application Use Cases associated with the Application.
- The *Manage* tab allows enabling Project Management features in the PRPC Developer portal for developer using this application.

4. Enterprise Application Accelerator (EAA). While this wizard has been around for a long time, it now cleanly dovetails into the output of the Application Profiler wizard. The EAA can create a complete Enterprise Class Structure consisting of a Customer-Built Framework (CBF) on the Framework layer and organizational implementation classes on the Implementation layer.

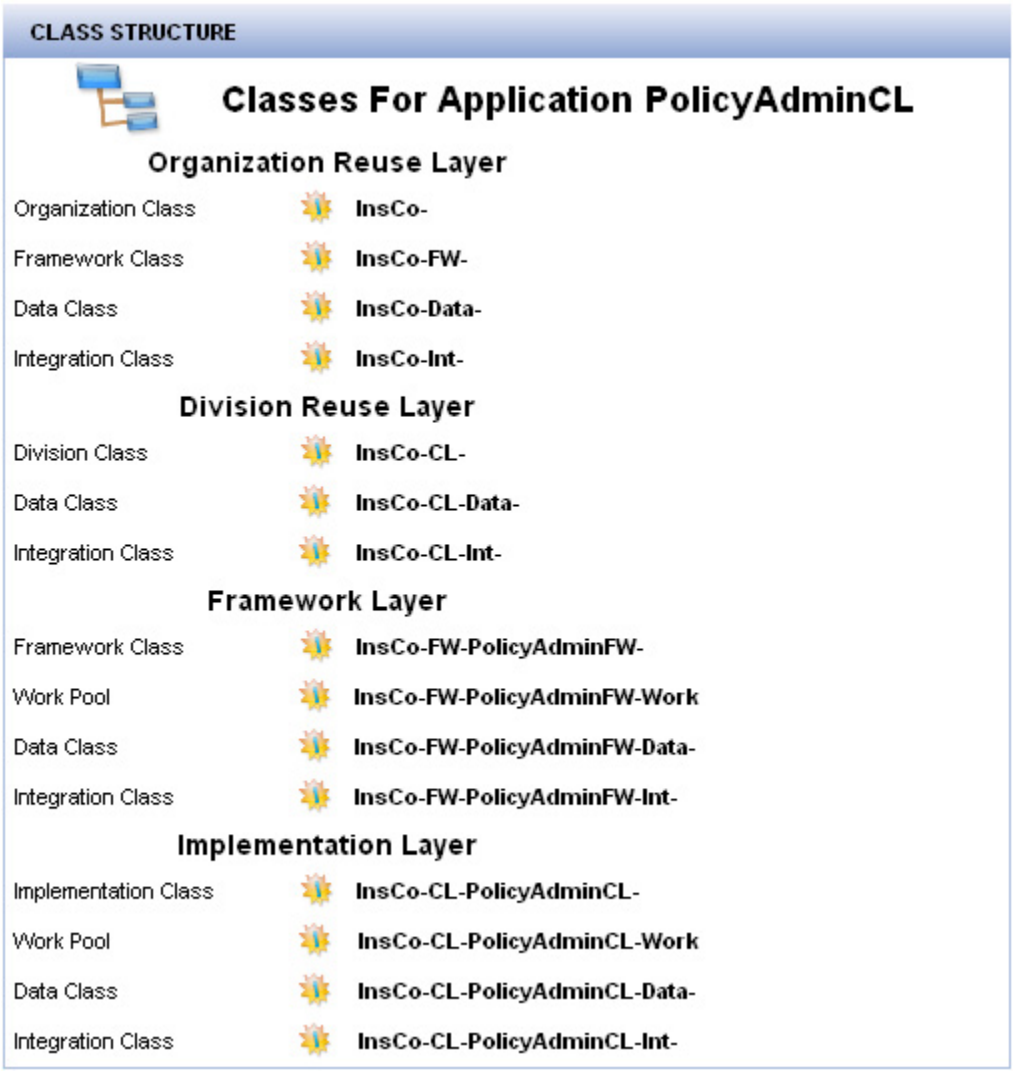

Figure 3: Sample ECS created by the Enterprise Application Accelerator

These enhancements have prompted an evolution of the best practices around enterprise reuse. This is the reason why some of the sections of this document break out best practices based on whether the application is being developed on a PRPC v5.4 vs. a pre-5.4 platform.

3 Implementing Applications Based on The Enterprise Class Structure

The following sections contain practical guidelines for implementing applications based on the Enterprise Class Structure. The guidelines use the layer naming of the PRPC v5.4 Enterprise Class Structure. However, they are applicable for the pre-5.4 Enterprise Class Structure as well.

3.1 In Which Layer Do You Put New Rules

In order for a Customer Built Framework (CBF) in the Framework layer to be reusable, the CBF needs to "stand on its own", it must contain all the necessary rules to run them even without an Implementation layer.

> ***Note:*** *Toward that end, many customers initially create only the CBF class hierarchy and develop and test the application in that class layer. Once the application is up and running, they create the Implementation layer, which contains mostly "stub" flows and activities. They add specialized rules as needed, by opening the generalized counterpart in the CBF and saving a copy of the rule to the class & RuleSet in the Implementation layer.*

When you are starting from a "green field", you are not reusing an existing CBF but you are building the CBF and the Implementation layer of the application as part of your development, the question arises where to put a given rule: in the Framework layer or in the Implementation layer?

The answer is: either in the Framework layer, or in both in the Framework and Implementation layers. **A rule should never exist only in the Implementation layer because then the CBF cannot use it.** Whether the rule is just in the Framework layer or in both the Implementation and Framework layers depends if you consider the rule as generic and reusable, or as specialized. If you can't think of a generalized version, that is probably an indication that the "specialized" rule is actually generic and should be put into the Framework layer.

Again, your guiding principle should be to create a CBF that can stand (and run) on its own. This is a prerequisite for the CBF to be reused by other organizational divisions.

3.2 What To Do if a Rule is Put into the Wrong Layer

If you followed the above recommendation, you will either create a generic and a specialized version of a given rule, or you will create only a generic version of that rule. If you later find out that the generic rule is actually wrong, at a time when the (shareable) generic application RuleSet is already locked, you can "withdraw" the rule by marking its availability in a higher RuleSet as "Withdrawn". Please see the PRPC online help for more information on this option.

3.3 Calling Flows in the Framework Layer

Even if your application is using a CBF as the basis, the work objects are actually created in the work class of the organizational layer. This is where you have to do your mapping of the class group to a database table using an Admin Database Table instance.

The list of flows that appear in the New gadget of the work portal, or the Run > Process dropdown of the Developer portal is dynamically built from flow rules available in the organizational work classes. Therefore, you will need to create a stub flows in the respective work class that calls the generic flow rule in the Framework layer.

The stub flows must have a name different from the name of the corresponding generic flow. Choose a name for the specialized flow that contains the name of the generic flow for easier reference, such as *CreateCaseStart* [specialized] and *CreateCase* [generalized].

3.4 Activities

Activities that create work objects (such as agent activities or service activities) must be defined in the Implementation layer. These activities should create a pyWorkPage of the Implementation layer work class and then call the respective activity in the Framework layer.

The stub activities must have a name different from the name of the corresponding generic activity. Choose a name for the specialized activity that contains the name of the generic activity for easier reference, such as *CreateCaseSvcStart* [specialized] and *CreateCaseSvc* [generalized].

3.5 Connector Activities and Connect Rules

Connector activities, activities that call Connect rules (such as Connect SOAP) should be created in the Framework layer. The Connect rules themselves should retrieve the values of environment properties, such as endpoint URL, server name, request queue, response queue, etc. from Global Resource settings.

Using the syntax *=MyDeclarativePage.pySOAPURL* instead of a hard-coded URL string allows dynamic settings of these environment properties in Global Resource Settings. The entire mechanism is explained in PDN article http://pdn.pega.com/Devnet/PRPCv5/KB/24171.asp.

Some types of Connect rules (Connect SOAP, Connect dotNet, Connect HTTP) allow you to include authentication credentials in HTTP headers. These credentials cannot be stored in Global Resource settings. The respective username and password properties must be set in a wrapper activity in the Implementation layer that calls the Connect activity in the Framework layer.

3.6 Services

The Service Package name must be a system-wide unique name; therefore, service packages with unique names must be created for each Implementation layer. The Service Rules contain the Applies To class name of the Service Activity. In most cases this class must be the Implementation work class. Therefore, you need to create clones of Service Rules in the Implementation work class.
Because Service Rules have the package name as the database key, the implementation specific Service Package must be created first.

3.6.1 Service Package

The Service Package (a data instance) must be created separately for each Implementation layer using a unique identifier name that starts with a letter and use only alphanumeric and dash characters.
We recommend creating a **template** of the Service Package that can be cloned (Save As) for each Implementation layer. The Service Access Group(Context tab) needs to be updated to reference the application specific access group.

3.6.2 Service Rule

Service Rules must be copied to the Implementation layer because they reference the class of the Service Activity which in most cases must be an Implementation layer work class. We recommend creating **template** Service Rules on the Framework layer that can be cloned for each Implementation layer. The developer opens the Service Rule template and saves it to an application RuleSet using the application specific Service Package identifier. The Primary Page Class (Service tab) needs to be updated to reference the respective work class in the Implementation layer.

3.6.3 Deployment

The Deployment files (WSDL for Service SOAP or dotNet, JAR for Service EJB or Service Java) contain the endpoint URL and must therefore be generated on each implementation node the application is deployed. Once all Service Rules have been created the developer should open the Service Package again and click the Show Service Methods button in the Methods tab to list the services that have the

package name as the first key part. To generate the deployment files the developer must press the Generate Deployment Files button in the Deployment tab.

3.7 How to Test the CBF

As mentioned earlier, a CBF must stand and run on its own. To test a CBF, you will need to create some infrastructure:

- Database Table instance
- Access Roles
- Access Groups
- Organization Structure
- Operator IDs with the requisite access
- Workbaskets
- Work groups

Exercise care to prevent name collisions with the Implementation layer. It is recommended that you prefix the name of test data instances with the string 'CBF' followed by the framework name.

3.8 Versioning of External Services (Any Protocol)

How do I re-generate a connector if the new service version does not provide all of the original methods / attributes of the previous version?

What about the proxy classes (Axis client stub classes) in PRPC prior to version 5.4 – can these classes be unique, i.e. are they located in different directories / packages? Yes they are in a generated directory / package.

We recommend eliminating the risk of backward incompatibility by using different service names for different versions of services.

3.9 Specialization of a CBF

Can CBFs be specialized within the Framework layer? Yes, consider using leaf classes of the Framework layer class. For example, to specialize the *MyCo-FW-InvgApp-Work* work class you could create two specialized leaf classes: *MyCo-FW-InvgApp-Work-Investigate* and *MyCo-FW-InvgApp-Work-ReqCopy*.

3.10 Making a Single Work Object Available as Cover or Object

Occasionally you may run into the requirement to use a CBF as a work cover object in one specialized instantiation, and as a plain work object in another. To do that, add two leaf classes to the work class: the first directly or indirectly inherits from Work-Object- and the second from Work-Cover-.

All rules that make up the application are located in the parent work class; the leaf classes provide mere entry points that add the standard Work-Cover- and Work-Object- rules from the PRPC product layer. The specialized work class in the organizational layer may inherit from either the work cover or work object leaf class.

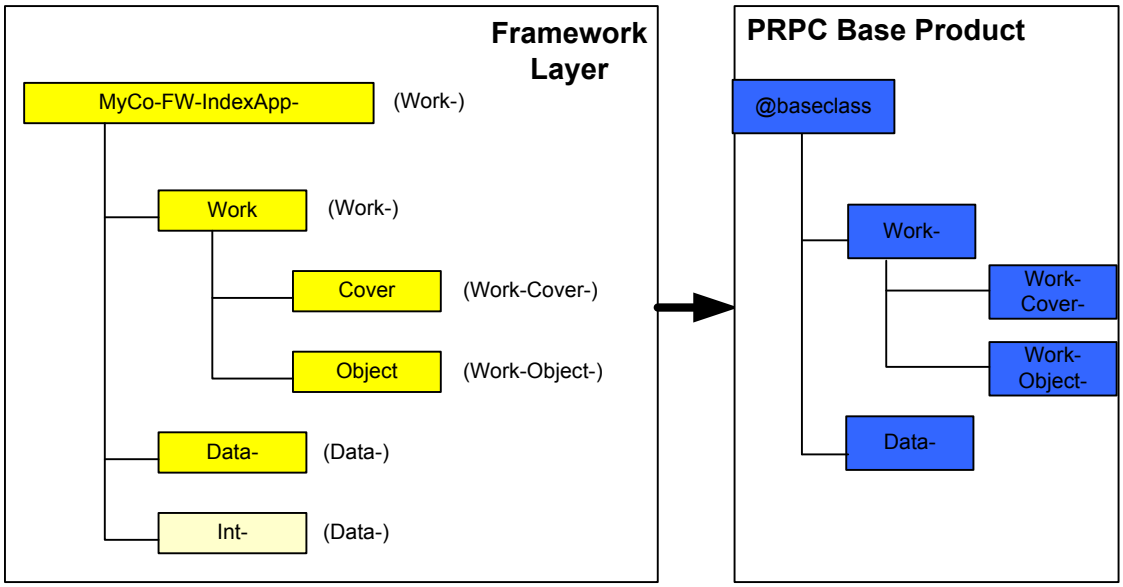

Figure 4: Making a single work class available as Cover and as Work Object

3.11 RuleSets

The rules of an application, shared or specialized, are stored in several RuleSets. There exists at least one RuleSet per layer of the Enterprise Class Structure. Since PRPC v5.4, the RuleSet rule form allows you to categorize RuleSets into Standard, Component, Shared and Override RuleSets. The RuleSets that contain the rules in the various layers must be categorized as shown in Table 4 below.

Enterprise Class Structure Layer (PRPC 5.4)	Enterprise Class Structure Layer (PRPC versions before 5.4)	RuleSet Type (PRPC 5.4)
Implementation Layer	Specialized Organizational Layer (ORG)	Standard
Framework Layer Customer-Built Framework (CBF)	Generalized Application Layer (GAL)	Standard (after all, a CBF is an application by itself)
Component (embedded property of a Work class; reuse through composition rather than inheritance)	Component (embedded property of a Work class)	Component
Enterprise Reuse Layer Divisional Reuse Layer	Enterprise Shared Layer (ESL)	Standard (these RuleSets typically contain classes and their rules. Therefore, they do not meet the requirements for RuleSet type Shared)

Table 4: Enterprise Class Structure Layers and RuleSet Categories

3.12 Applications

The RuleSets and Use Cases of the specialized application (Implementation layer) should be defined in Application rules. Specialized applications are built on the framework applications.

The RuleSets and Use Cases of customer built frameworks (Framework layer) should be defined in separate Application rules. The framework applications are built on either another CBF, a Pega built framework application, or the PRPC 5.4 application.

Extensions (enterprise reuse and divisional reuse layers) **should not** have their own Application rules as they do not constitute complete applications. Nevertheless, their RuleSets must be designated as type *Standard* in the Category tab of the RuleSet form because its rules don't operate on a top-level class, but rather contains a series of classes and their rules. Extension RuleSets should be referenced in the Application RuleSets list of the Application rules of the frameworks and/or specialized applications that depend on them.

3.13 Access Roles

Typically, access roles need to be created on the Framework layer only as security is inherited from the parent classes in the Framework to the descendent classes on the Implementation layer.

We recommend creating application specific access roles by cloning and customizing the Pega OOTB access roles using the Access Role Editor tool, available on the developer portal by selecting Tools >

Security > Role Names. Create separate access roles for each user profile, for example: ClaimsApp:ClaimsRep, ClaimsApp:ClaimsMgr, ClaimsApp:SysAdmin, etc.

When you need to override access security for an Implementation work class you may create an access role just for that work class in an application RuleSet. This access role can be referenced in the user's access group in addition to the access role from the framework. The access settings that you define on the Implementation work class will override the respective settings of its parent class as defined in the framework access role.

3.14 Product and Product Patch

Similar to Application rules, Product rules should be defined to package the RuleSets and data instances for the Implementation and the Framework layers to make their rules available for migration.

You should define one Product rule for each Implementation layer; you should define one Product rule for each framework; you may define Product rules for the extensions.

4 Appendix A: Compatibility of Applications and Frameworks built on the pre-5.4 Enterprise Class Structure

This section documents considerations and caveats for applications currently in production that are built on the recommended pre-5.4 Enterprise Class Structure.

4.1 Applications

Applications built on the pre-5.4 Enterprise Class Structure will continue to exhibit their reusability aspects without changes to the class structure when they are migrated to PRPC 5.4.

4.2 Customer-Built Frameworks (CBFs)

CBFs built on the pre-5.4 Enterprise Class Structure can be reused by 5.4 applications as is. There is no need to re-factor these applications or CBFs to conform to the new class naming conventions, i.e. using MyCo-FW- as the namespace qualifier.

In new versions of a CBF you may want to consider changing the directed parent of the CBF work and data classes to Work- and Data-, respectively. (This requires that you first copy extension rules from the Shared-Work- and Shared-Data- classes to the Work- and Data- classes, respectively.)

4.3 Extensions

To support applications built on the PRPC v5.4 enterprise class hierarchy, and new versions of CBFs (see above) you need to copy all extension rules from the Shared-Work- and Shared-Data- classes to the MyCo- [directed inheritance from Work-] and MyCo-Data- [directed inheritance from Data-] classes, respectively.)

4.4 Data and Interface Classes

Subclasses of Shared-Data- and Shared-IF- contain data and interface classes that are shared enterprise-wide. They are typically referenced directly in the Pages and Classes tab. The class structures and the rules defined in them can remain as is.

Newly created enterprise-wide or division-wide data and generated interface classes should follow the new naming conventions. Their base classes should comply with the following recommendations:

Enterprise-wide classes:

MyCo-Data- (Data-)

MyCo-Int- (Data-)

Division-wide base classes:

MyCo-Division1-Data- (Data-)

MyCo-Division1-Int- (Data-)

5 Appendix B: Classes & RuleSets generated by the Enterprise Application Accelerator

Class	RuleSet	Direct Parent Class
MyCo-	MyCo	Work-
MyCo-FW-	MyCo	Work-
MyCo-Data-	MyCo	Data-
MyCo-Int-	MyCoInt	Data-
MyCo-Div1-	MyCoDiv1	Work-
MyCo-Div1-Data-	MyCoDiv1	Data-
MyCo-Div1-Int	MyCoDiv1Int	Data-
MyCo-FW-FrameworkName1-	FrameworkName1	Work-
MyCo-FW-FrameworkName1-Work	FrameworkName1	Work-
MyCo-FW-FrameworkName1-Work-Type1	FrameworkName1	Work-Object-
MyCo-FW-FrameworkName1-Work-Type2	FrameworkName1	Work-Object-
MyCo-FW-FrameworkName1-Data-	FrameworkName1	Data-
MyCo-FW-FrameworkName1-Int-	FrameworkName1Int	Data-
MyCo-Div1-Implementation1-	MyCoImplementation1	Work-
MyCo-Div1-Implementation1-Work	MyCoImplementation1	MyCo-FW-FrameworkName1-Work
MyCo-Div1-Implementation1-Work-WorkType1	MyCoImplementation1	MyCo-FW-FrameworkName1-Work-WorkType1
MyCo-Div1-Implementation1-Work-WorkType2	MyCoImplementation1	MyCo-FW-FrameworkName1-Work-WorkType2
MyCo-Div1-Implementation1-Data-	MyCoImplementation1	Data-
MyCo-Div1-Implementation1-Int-	MyCoImplementation1Int	Data-

RuleSet	Prerequisite RuleSet(s)
MyCoImplementation1	FrameworkName1 MyCoDiv1 MyCoImplementation1Int
MyCoImplementation1Int	FrameworkName1Int MyCoDiv1Int
MyCoDiv1	MyCo MyCoDiv1Int
MyCoDiv1Int	MyCoInt
MyCo	MyCoInt
MyCoInt	Pega-AppDefinition
FrameworkName1	Pega-AppDefinition FrameworkName1Int
FrameworkName1Int	MyCo